

Infrastructure Automation for Containerized Applications on AWS EKS Using Terraform and GitHub Actions

Ayomide Ajayi

larry.ayomide@techbleat.co.uk

Project Overview

This project presents a comprehensive, automated solution for deploying containerized applications on Amazon Elastic Kubernetes Service (EKS) using Infrastructure as Code (IaC) principles. Leveraging Terraform for environment provisioning and GitHub Actions for orchestrating CI/CD pipelines, it delivers a scalable, secure, and production-ready Kubernetes infrastructure.

The deployment architecture includes a custom virtual private cloud (VPC) with three-node EKS clusters spanning two Availability Zones, ensuring high availability and fault tolerance. Infrastructure state management is handled securely using an S3 backend, with DynamoDB providing state locking to support safe, concurrent operations. Declarative Kubernetes manifests are applied post-provisioning to deploy a web application, accessible via an external LoadBalancer service.

The pipeline is structured into modular stages—provisioning, application delivery, and teardown—allowing for repeatable, auditable workflows. A conditional teardown mechanism, activated via manual workflow dispatch (`auto_destroy=true`), safely decommissions infrastructure after a short delay, supporting both ephemeral and persistent environments. By embracing cloud-native design patterns and automating the full deployment lifecycle, this project advances DevOps maturity, reduces manual overhead, and enhances the reproducibility of infrastructure and application deployment processes.

Project's Step-by-Step Implementation

1. Push the Image to AWS ECR (*Manually done outside the pipeline but can be integrated if required*)

1.1 Authenticate Docker to ECR:

```
aws ecr get-login-password --region eu-west-2 | docker login --username AWS --password-stdin <your_aws_account_id>.dkr.ecr.eu-west-2.amazonaws.com
```

1.2 Create the ECR Repository (if not already created):

```
aws ecr create-repository --repository-name ghs-nginx-app46 --region eu-west-2
```

1.3 Tag Your Local Image:

```
docker tag ghs-nginx-app46:latest < your_aws_account_id >.dkr.ecr.eu-west-2.amazonaws.com/ghs-nginx-app46:latest
```

1.4 Push the Image to ECR:

```
docker push < your_aws_account_id >.dkr.ecr.eu-west-2.amazonaws.com/ghs-nginx-app46:latest
```

2. Initialize Terraform project

- Define provider "aws" with region & credentials.
- Set terraform { backend "s3" { ... } } for remote state in S3, add DynamoDB table for locking. (*Backend manually created*)
- **Remote state was used for getting VPC outputs via**
`terraform_remote_state`

3. Network configuration and provisioning

- Provisioned custom VPC via Terraform, with state managed remotely in an S3 backend for consistency and collaboration.
- Optionally include VPC Endpoint for EKS or S3.

4. EKS cluster provisioning and Managed Node Group

A fully managed EKS control plane and node group are deployed using only Terraform core resources, enabling fine-grained control over IAM roles, subnet associations, and scaling parameters—without relying on third-party modules.

- Create EKS control plane.
- Define worker Node Group: 3 nodes (managed node groups) across private subnets in 2 availability zones (AZs).
- Configure IAM policies and roles amongst others.

5. GitHub Actions CI/CD setup

- Stored Terraform + Kubernetes manifests in a repo.
- Workflow triggers on pushes to main:
 1. Checkout code.
 2. Use hashicorp/setup-terraform.
 3. terraform init, plan, apply (with stored credentials).
 4. Install kubectl && Use aws eks credentials to configure kubectl.
 5. verifies context authentication and node readiness.
 6. Deployed deployment and load balancer service
 7. Post-deployment checks validate LoadBalancer endpoint
 8. Monitor rollout progress to confirm application stability

6. Creation of essential IAM policies including but not limited to:

- Creation of the Node IAM Role
- Creation of an EKS IAM Role for the Control Plane to manage AWS resources.
- Ensure secure GitHub Actions secrets (such as AWS keys).

7. Cost optimisation

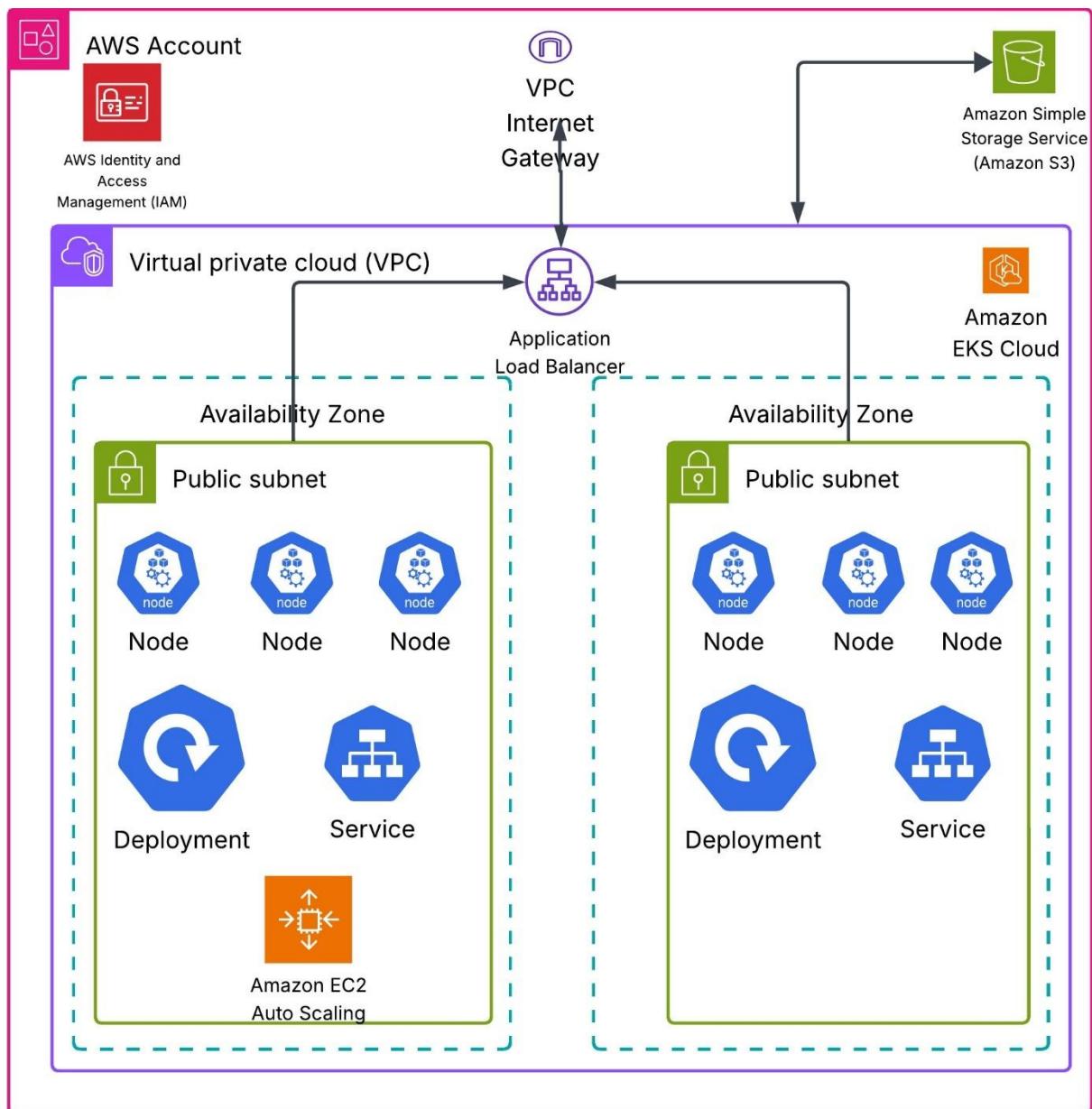
- Implemented a conditional teardown stage gated by a manual workflow_dispatch input (auto_destroy=true).
- Introduces delay before safely destroying the EKS cluster and VPC via Terraform, offering flexibility for ephemeral environments or controlled cleanup.

8. Validation & Monitoring

- Validate cluster health and node automating.
- Confirm app via LoadBalancer endpoint.

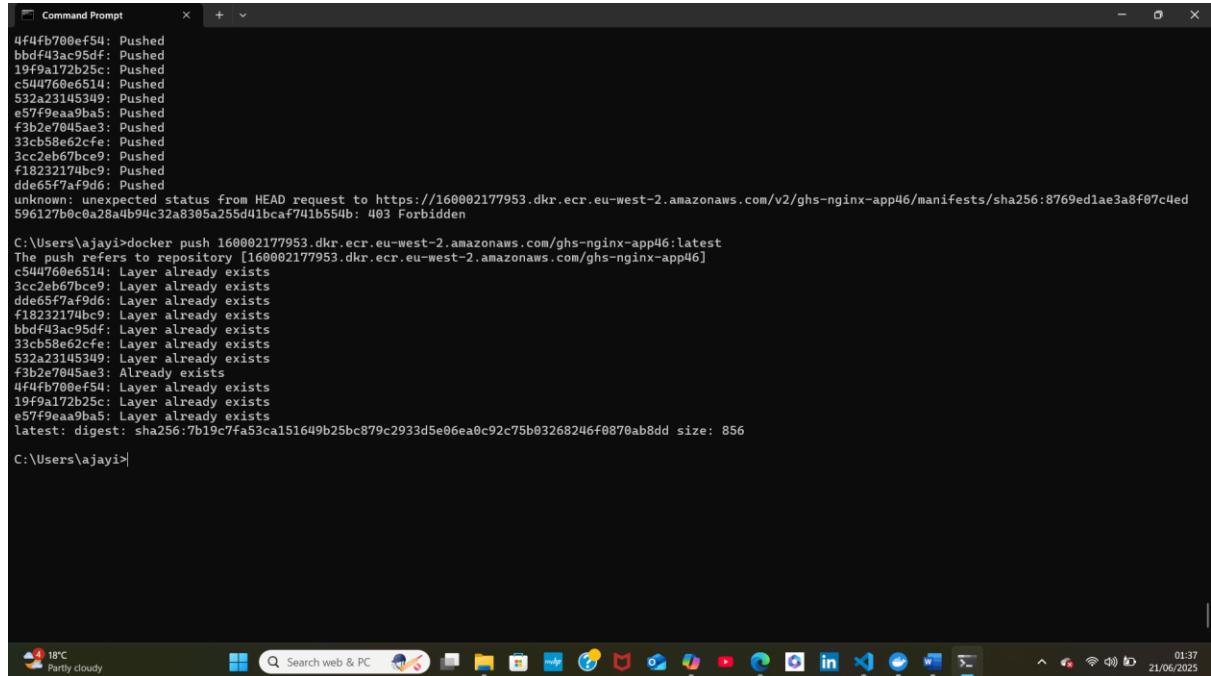
Overall, this project demonstrates proficiency in cloud-native tooling and DevOps principles, highlighting my expertise in AWS IAM, Terraform-based infrastructure-as-code, Kubernetes deployment pipelines, and GitHub Actions for CI/CD automation—all wrapped in a secure, observable, and declarative workflow.

Project Overview Diagram- High-Level Architecture



Project Artefacts (a few)

Image Pushed to AWS ECR

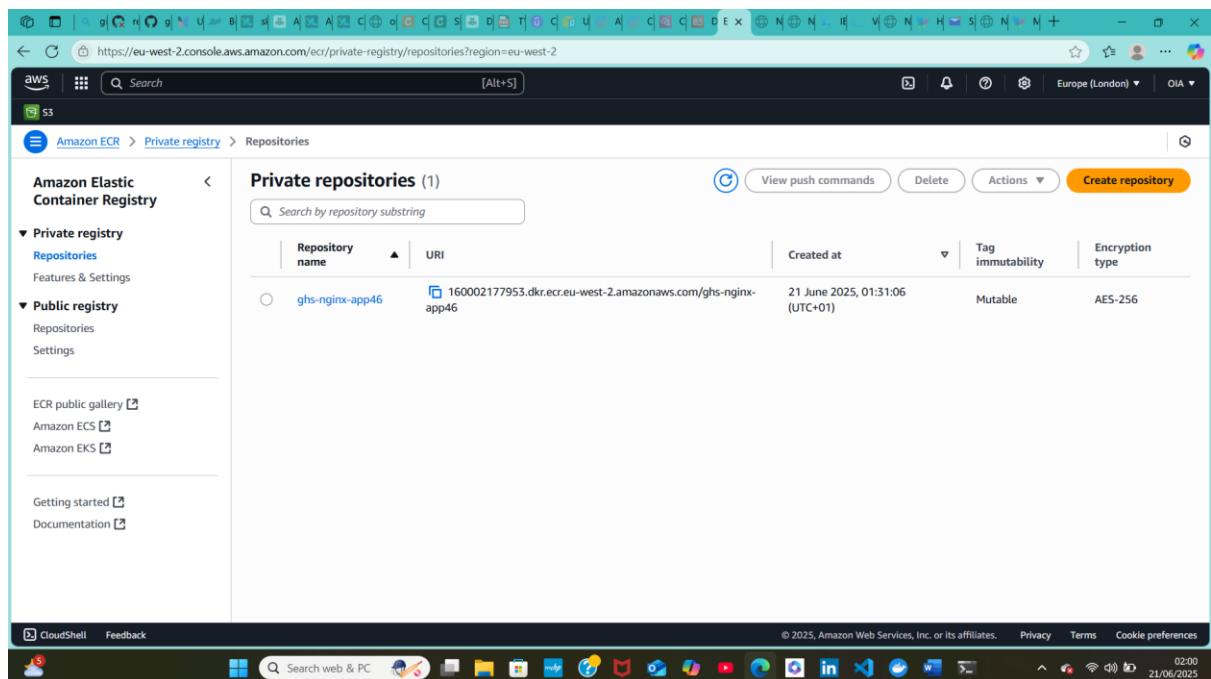


```
4f4fb798ef54: Pushed
bbdf43ac95df: Pushed
19f9a172b25c: Pushed
c54f766e6514: Pushed
532a23145349: Pushed
e57f9eaa9ba5: Pushed
f3b2e7045ae3: Pushed
33cb58e62cf6: Pushed
3cc2e6b7bc9: Pushed
f18232174bc9: Pushed
ddee5f7af9d6: Pushed
unknown: unexpected status from HEAD request to https://160002177953.dkr.ecr.eu-west-2.amazonaws.com/v2/ghs-nginx-app46/manifests/sha256:8769ed1ae3a8f07c4ed596127b0c0a28a4b94c32a8305a255d41bca741b554b: 403 Forbidden

C:\Users\ajayi>docker push 160002177953.dkr.ecr.eu-west-2.amazonaws.com/ghs-nginx-app46:latest
The push refers to repository [160002177953.dkr.ecr.eu-west-2.amazonaws.com/ghs-nginx-app46]
c54f766e6514: Layer already exists
3cc2e6b7bc9: Layer already exists
ddee5f7af9d6: Layer already exists
f18232174bc9: Layer already exists
bbdf43ac95df: Layer already exists
33cb58e62cf6: Layer already exists
532a23145349: Layer already exists
f3b2e7045ae3: Already exists
4f4fb798ef54: Layer already exists
19f9a172b25c: Layer already exists
e57f9eaa9ba5: Layer already exists
latest: digest: sha256:7b19c7fa53ca151649b25bc879c2933d5e06ea0c92c75b03268246f0870ab8dd size: 856

C:\Users\ajayi>
```

Elastic Container Registry



The screenshot shows the AWS ECR console interface. On the left, there's a navigation sidebar with options like 'Amazon Elastic Container Registry', 'Private registry', 'Public registry', and links to 'ECR public gallery', 'Amazon ECS', 'Amazon EKS', 'Getting started', and 'Documentation'. The main content area is titled 'Private repositories (1)' and lists a single repository named 'ghs-nginx-app46'. The table includes columns for 'Repository name', 'URI', 'Created at', 'Tag immutability', and 'Encryption type'. The repository details are: URI - 160002177953.dkr.ecr.eu-west-2.amazonaws.com/ghs-nginx-app46, Created at - 21 June 2025, 01:31:06 (UTC+01), Tag immutability - Mutable, Encryption type - AES-256.

Repository name	URI	Created at	Tag immutability	Encryption type
ghs-nginx-app46	160002177953.dkr.ecr.eu-west-2.amazonaws.com/ghs-nginx-app46	21 June 2025, 01:31:06 (UTC+01)	Mutable	AES-256

The screenshot shows the Amazon ECR console interface. On the left, a sidebar navigation includes 'Amazon Elastic Container Registry', 'Private registry' (selected), 'Public registry', and links to 'ECR public gallery', 'Amazon ECS', and 'Amazon EKS'. The main content area displays a table titled 'Images (3)' with columns for 'Image tag', 'Artifact type', 'Pushed at', 'Size (MB)', 'Image URI', 'Digest', and 'Last recorded pull time'. The table lists three entries: 'latest' (Image index, pushed 21 June 2025, size 22.18 MB, digest sha256:7b19c7fa53ca151...), a second 'Image' entry (pushed 21 June 2025, size 22.18 MB, digest sha256:8769ed1ae3a8f0...), and a third 'Image' entry (pushed 21 June 2025, size 0.00 MB, digest sha256:6bf2fb2b24f3dbf9...). A message at the top states: 'Image scan overview, status, and full vulnerabilities are now displayed in the image detail page. To access, click an image tag.' Buttons for 'Delete', 'Details', 'Scan', and 'View push commands' are available at the top right.

IAM PERMISSIONS

The screenshot shows the AWS IAM console. The left sidebar includes 'Identity and Access Management (IAM)', 'Access management' (User groups selected), 'Access reports', and 'Access advisor'. The main content area shows the 'Permissions' tab for a group named 'DevOps-2025'. It displays a list of 'Permissions policies (8)' with columns for 'Policy name', 'Type', and 'Attached entities'. Policies listed include 'AmazonEC2FullAccess', 'AmazonEKSNetworkingPolicy', 'AmazonS3FullAccess', 'AmazonS3TablesFullAccess', 'AmazonSSMManagedInstanceCore', 'AmazonVPCFullAccess', 'ecr-policy' (Customer inline), and 'eks-policy' (Customer inline). A 'Filter by Type' dropdown is set to 'All types'. Buttons for 'Simulate', 'Remove', and 'Add permissions' are at the top right. A note at the bottom says 'You can attach up to 10 managed policies.'

IAM Roles

The screenshot shows the AWS IAM Roles page with 11 entries listed:

Role name	Trusted entities	Last activity
AWSServiceRoleForAmazonEKS	AWS Service: eks (Service-Linked Role)	14 minutes ago
AWSServiceRoleForAmazonEKSNodegroup	AWS Service: eks-nodegroup (Service-Linked Role)	9 minutes ago
AWSServiceRoleForAutoScaling	AWS Service: autoscaling (Service-Linked Role)	13 minutes ago
AWSServiceRoleForCloudWatchAlarms_ActionSSM	AWS Service: ssm.alarms.cloudwatchalarms (Service-Linked Role)	-
AWSServiceRoleForElasticLoadBalancing	AWS Service: elasticloadbalancing (Service-Linked Role)	24 minutes ago
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
EC2-CloudWatch-Role	AWS Service: ec2	-
EC2Monitoring	AWS Service: ec2	-
eks-cluster-role	AWS Service: eks	8 minutes ago
eks-node-role	AWS Service: ec2	13 minutes ago

S3 Backend

The screenshot shows the AWS S3 Buckets page for the 'my-eksproject-store' bucket. The 'env/' folder contains two objects:

Name	Type	Last modified	Size	Storage class
eks/	Folder	-	-	-
vpc/	Folder	-	-	-

The screenshot shows the AWS S3 console interface. The left sidebar lists 'General purpose buckets' and 'Storage Lens'. The main area shows the contents of the 'vpc/' folder in the 'env/vpc' bucket. There is one object named 'terraform.tfstate' which is a tfstate file. The table below provides details about this object.

Name	Type	Last modified	Size	Storage class
terraform.tfstate	tfstate	June 22, 2025, 11:17:30 (UTC+01:00)	14.1 KB	Standard

The screenshot shows the AWS S3 console interface. The left sidebar lists 'General purpose buckets' and 'Storage Lens'. The main area shows the contents of the 'eks/' folder in the 'env/eks' bucket. There is one object named 'terraform.tfstate' which is a tfstate file. The table below provides details about this object.

Name	Type	Last modified	Size	Storage class
terraform.tfstate	tfstate	June 22, 2025, 19:14:33 (UTC+01:00)	14.9 KB	Standard

Custom VPC

VPC Details:

- VPC ID:** vpc-03882348bddab8ab3
- State:** Available
- Tenancy:** default
- Main network ACL:** acl-00da8361cf058217d
- IPv6 CIDR (Network border group):** -
- Default VPC:** No
- Network Address Usage metrics:** Disabled
- Block Public Access:** Off
- DNS hostnames:** Enabled
- DHCP option set:** dopt-03b1339c18606202a
- IPv4 CIDR:** 192.168.0.0/16
- Route 53 Resolver DNS Firewall rule groups:** -
- Main route table:** rtb-09a3cd1fb1395b76
- IPv6 pool:** -
- Owner ID:** 160002177953

Resource map:

- VPC:** Show details
- Subnets (4):** Subnets within this VPC
- Route tables (2):** Route network traffic to resources
- Network interfaces (1):** Connect to EC2 instances

Subnets

Subnets (5) Info

Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR
pubsub-1	subnet-0eb2d54b76b1880a1	Available	vpc-03882348bddab8ab3 KubeVPC	Off	192.168.1.0/24
pubsub-2	subnet-0152d5ae4b5ee2a84	Available	vpc-03882348bddab8ab3 KubeVPC	Off	192.168.2.0/24
privsub-1	subnet-09f2337b6dddfaf3c3	Available	vpc-03882348bddab8ab3 KubeVPC	Off	192.168.3.0/24
-	subnet-0f5c86cf7a5e4c72d	Available	vpc-01adff1fc36b61af3a	Off	172.31.32.0/20
privsub-2	subnet-0f3c97e5969fa84a	Available	vpc-03882348bddab8ab3 KubeVPC	Off	192.168.4.0/24

Route tables

The screenshot shows the AWS VPC Route Tables page. On the left, there's a navigation sidebar with sections like VPC dashboard, EC2 Global View, Virtual private cloud (Your VPCs, Subnets, Route tables), Internet gateways, Egress-only Internet gateways, Carrier gateways, DHCP option sets, Elastic IPs, Managed prefix lists, NAT gateways, Peerings, Security (Network ACLs, Security groups), and CloudShell/Feedback. The main area has a header "Route tables (1/3) Info" with a search bar and filters. It lists three route tables: "PublicRouteTable" (selected), "rtb-09a3dcdf1b1395b76", and "rtb-0459992bf21cfdec6". Each row includes columns for Name, Route table ID, Explicit subnet associations, Edge associations, Main, VPC, and Owner. Below this, a detailed view for "rtb-0722997910e47b6b4 / PublicRouteTable" is shown with tabs for Details, Routes, Subnet associations, Edge associations (selected), Route propagation, and Tags. The Details tab shows the Route table ID (rtb-0722997910e47b6b4), Main (No), VPC (vpc-03882348bddab8ab3 | KubeVPC), and Owner ID (160002177953). The Edge associations tab shows 2 subnets: "subnet-0eb2d54b76b1880a1 / pubsub-1" and "subnet-0152d5ae4b5ee2a84 / pubsub-2".

Cluster IGW

The screenshot shows the AWS VPC Internet Gateways page. The left sidebar is identical to the previous screenshot. The main area has a header "igw-0dcdd3d372726f185f / ClusterIGW" with a Details tab selected. It shows the Internet gateway ID (igw-0dcdd3d372726f185f), State (Attached), VPC ID (vpc-03882348bddab8ab3 | KubeVPC), and Owner (160002177953). Below this is a Tags section with a table showing a single tag: Name (ClusterIGW). There are tabs for Details, Info, Routes, Subnet associations, Edge associations, Route propagation, and Tags.

EKS Cluster

The screenshot shows the AWS EKS Cluster list page. On the left, there's a sidebar with navigation links for Amazon Elastic Kubernetes Service, Settings, Amazon EKS Anywhere, and Related services. The main area displays a table titled 'Clusters (1) Info' with one row for 'laredo-cluster'. The table columns include Cluster name, Status, Kubernetes version, Support period, Upgrade policy, Created, and Provider. The cluster details are: laredo-cluster, Active, 1.29, Extended support until March 23, 2026, Extended, 2 hours ago, and EKS. There are buttons for 'Delete' and 'Create cluster' at the top right of the table.

The screenshot shows the AWS EKS Cluster details page for 'laredo-cluster'. The left sidebar is identical to the previous screenshot. The main content area has a prominent blue banner at the top stating: 'Your current IAM principal doesn't have access to Kubernetes objects on this cluster. This may be due to the current user or role not having Kubernetes RBAC permissions to describe cluster resources or not having an entry in the cluster's auth config map.' Below this, there's a 'laredo-cluster' summary card with buttons for 'Delete cluster', 'Upgrade version', and 'Monitor cluster'. A note below the card says: 'End of extended support for Kubernetes version (1.29) is March 23, 2026. If you don't upgrade your cluster to a later version before that date, it will be automatically upgraded to Kubernetes version 1.30.' The 'Cluster info' section shows the status as 'Active', Kubernetes version 1.29, a support period until March 23, 2026, and the provider as EKS. It also includes sections for Cluster health (green), Upgrade insights (3 green, 1 yellow), and Node health issues (0). At the bottom, there are tabs for Overview (selected), Resources, Compute, Networking, Add-ons, Access, Observability, Update history, and Tags.

NodeGroup

The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed. The main area displays a table titled "Instances (3) Info" with the following data:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
laredo-nodes-1	i-08a2aeb16334ae3ab	Running	t3.small	3/3 checks passed	View alarms	eu-west-2a
laredo-nodes-2	i-0c413785898f17350	Running	t3.small	3/3 checks passed	View alarms	eu-west-2a
laredo-nodes-3	i-0202420eac50cc8db	Running	t3.small	3/3 checks passed	View alarms	eu-west-2b

Below the table, a dropdown menu says "Select an instance". The bottom right corner shows the date and time: 22/06/2025 20:02.

Load Balancer

The screenshot shows the AWS Load Balancers page for a Classic Load Balancer with the ARN a112b036067094434803b3a363dee8fd. The left sidebar is collapsed. The main area displays the "Details" section with the following information:

Load balancer type	Status	VPC	Date created
Classic	3 of 3 instances in service	vpc-03882348bddab8ab3	June 22, 2025, 19:26 (UTC+01:00)
Scheme	Hosted zone	subnet-0eb2d54b76b1880a1	eu-west-2a (euw2-az2)
	Internet-facing	subnet-0152d5ae4b5ee2a84	eu-west-2b (euw2-az3)

Below the table, there is a note about migration to a next-generation load balancer, a "Launch migration wizard" button, and a "Distribution of targets by Availability Zone (AZ)" section. The bottom navigation bar includes tabs for "Listeners", "Network mapping", "Security", "Health checks", "Target instances", "Monitoring", "Attributes", and "Tags". The bottom right corner shows the date and time: 22/06/2025 20:03.

Auto Scaling Group (ASG)

The screenshot shows the AWS EC2 Auto Scaling groups page. On the left, there's a navigation sidebar with sections like AMI Catalog, Elastic Block Store, Network & Security, Load Balancing, and Auto Scaling (with Auto Scaling Groups selected). The main area displays a table for Auto Scaling groups. One group is listed:

Name	Launch template/configuration	Instances	Status	Desired capacity	Min	Max
eks-laredo-nodes-8ecbcc66-6353-aacb-6d34-c75a99ad36de	eks-Becbcc66-6353-aacb-6d34-c75a99ac	3	-	3	2	5

Below the table, it says "0 Auto Scaling groups selected". At the bottom, there are links for CloudShell and Feedback, along with the standard Windows taskbar.

The screenshot shows the details page for the specific Auto Scaling group from the previous screenshot. The URL is https://eu-west-2.console.aws.amazon.com/ec2/home?region=eu-west-2#AutoScalingGroupDetails:id=eks-laredo-nodes-8ecbcc66-6353-aacb-6d34-c75a99ad36de;view=details. The page has a similar layout with a sidebar and a main content area. The main content includes a "Capacity overview" section and a "Launch template" section.

eks-laredo-nodes-8ecbcc66-6353-aacb-6d34-c75a99ad36de Capacity overview

Desired capacity	Scaling limits (Min - Max)	Desired capacity type	Status
3	2 - 5	Units (number of instances)	-

Date created
Sun Jun 22 2025 19:01:27 GMT+0100 (British Summer Time)

Launch template

Launch template	AMI ID	Instance type	Owner
lt-0c6c9d68d213f6764 eks-8ecbcc66-6353-aacb-6d34-c75a99ad36de	ami-03107e05d7003b91b	t3.small	arn:aws:sts::160002177953:assumed-role/AWSServiceRoleForAmazonEKSNode group/EKS

Version
1
Security groups
-
Security group IDs
[sg-0c81f864810bfd955](#)

At the bottom, it says "Create time" and "Sun Jun 22 2025 19:00:54 GMT+0100 (British Summer Time)". The page also includes standard AWS footer links for CloudShell, Feedback, and cookie preferences.

Screenshot of the AWS Cloud Console showing the Auto Scaling group details for 'eks-laredo-nodes-8ecbcc66-6353-aacb-6d34-c75a99ad36de'.

General Information:

AMI Catalog	eks-8ecbcc66-6353-aacb-6d34-c75a99ad36de	Security group IDs	role/AWSServiceRoleForAmazonEKSNodegroup/EKS
Elastic Block Store	Version 1	Storage (volumes) /dev/xvda	Create time Sun Jun 22 2025 19:00:54 GMT+0100 (British Summer Time)
Network & Security	Description -	Key pair name -	Request Spot Instances No
Load Balancing	View details in the launch template console		
Auto Scaling	Edit		

Network:

Availability Zones euw2-az2 (eu-west-2a) euw2-az3 (eu-west-2b)	Subnet ID subnet-0eb2d54b76b1880a1 subnet-0152d5ae4b5ee2a84	Availability Zone distribution Balanced best effort
---	--	---

Instance type requirements:

Instance type	vCPUs	Memory	Storage	Network performance	Weight
t3.small	2	2 GiB	EBS-Only	Up to 5 Gigabit	-

Tags:

Key	Value	Tag new instances
ekscluster-name	laredo-cluster	Yes
eksnodegroup-name	laredo-nodes	Yes
k8s.io/cluster-autoscaler/enabled	true	Yes
k8s.io/cluster-autoscaler/laredo-cluster	owned	Yes
kubernetes.io/cluster/laredo-cluster	owned	Yes

Screenshot of the AWS Cloud Console showing the Advanced configurations for the same Auto Scaling group.

Advanced configurations:

Instance scale-in protection Not protected from scale in	Termination policies AllocationStrategy, OldestLaunchTemplate, Oldestinstance	Maximum instance lifetime -	Service-linked role arn:aws:iam::160002177953:role/aws-service-role/autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling
Placement group -	Suspended processes -	Default cooldown 300	Default instance warmup Disabled

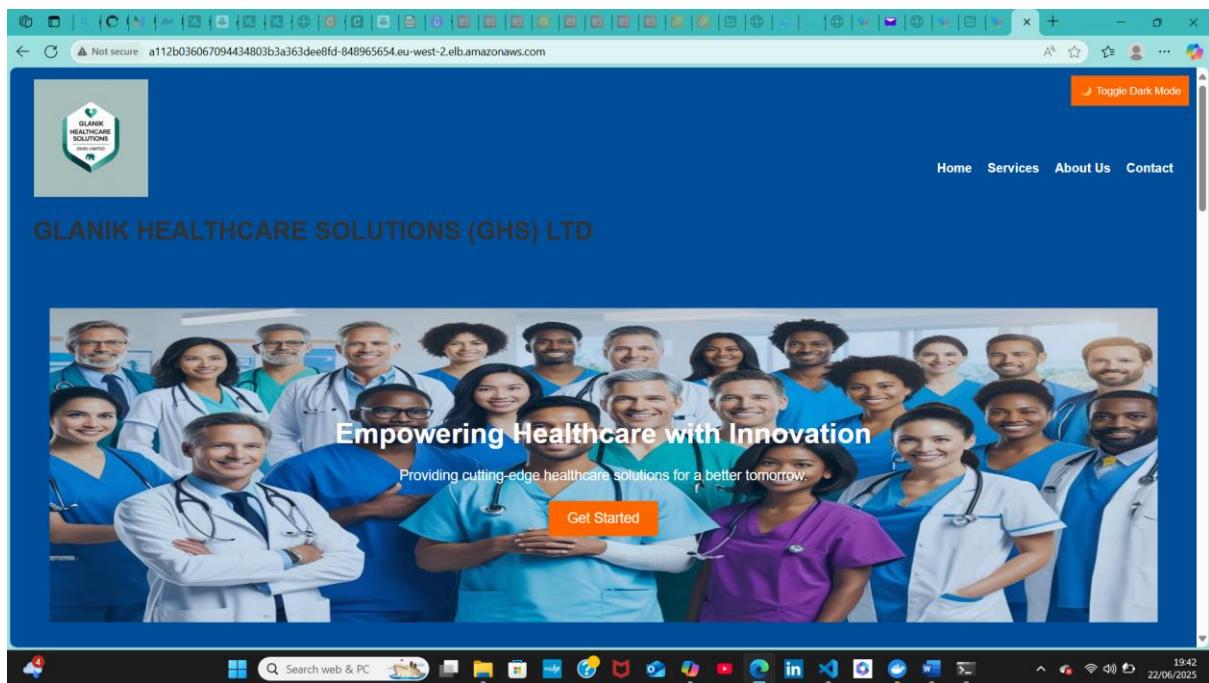
Tags:

Key	Value	Tag new instances
ekscluster-name	laredo-cluster	Yes
eksnodegroup-name	laredo-nodes	Yes
k8s.io/cluster-autoscaler/enabled	true	Yes
k8s.io/cluster-autoscaler/laredo-cluster	owned	Yes
kubernetes.io/cluster/laredo-cluster	owned	Yes

Loadbalancer Endpoint

LoadBalancer endpoint is available at: <http://a112b036067094434803b3a363dee8fd-848965654.eu-west-2.elb.amazonaws.com>

Web application Access via LoadBalancer Service Using the Endpoint above



A screenshot of the "About Us" page from the same website. The top navigation bar is visible, along with a "Toggle Dark Mode" button. The main content area features a heading "GHS was established in April 2022 as a consequence of the identified Healthcare workforce needs in the UK." Below this, there is a paragraph of text describing the organization's mission and expertise in providing personalized services across various healthcare domains. Further down, there is a section titled "Our Services" with three sub-sections: "Telemedicine Solutions" (showing a doctor video calling a patient), "Healthcare Data Analytics" (showing a stethoscope and digital icons), and "Medical Software Development" (showing a tablet displaying medical data). Each service section includes a brief description and a "Customized applications for healthcare" link at the bottom. The bottom of the screen shows a Windows taskbar with various pinned icons.

Not secure a112b036067094434803b3a363dee8fd-848965654.eu-west-2.elb.amazonaws.com

Toggle Dark Mode

Telemedicine Solutions Connecting patients with healthcare professionals remotely.	Healthcare Data Analytics AI-driven insights to optimize medical processes and improve patient care.	Medical Software Development Customized applications for healthcare management and patient tracking.
--	--	--



Healthcare Consultancy
Expert advice to enhance operational efficiency and compliance.



Healthcare Education
Promoting professional skills and competencies amongst Healthcare Practitioners.

19:44
22/06/2025

Not secure a112b036067094434803b3a363dee8fd-848965654.eu-west-2.elb.amazonaws.com

Toggle Dark Mode

What Our Clients Say

Client giving feedback
"GHS transformed our healthcare operations with their innovative solutions!"
- Healthcare Provider

Contact Us

Your Name:

Your Email:

Phone Number:

Subject:

Your Message:

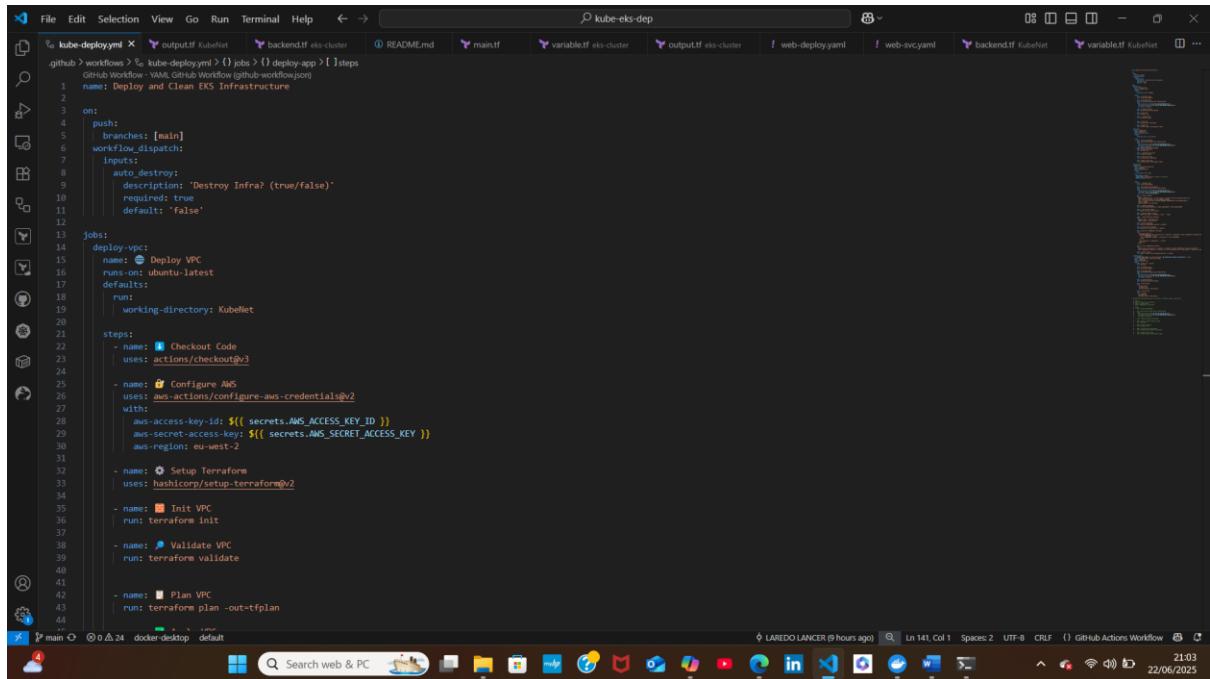
How would you rate our service?

Send

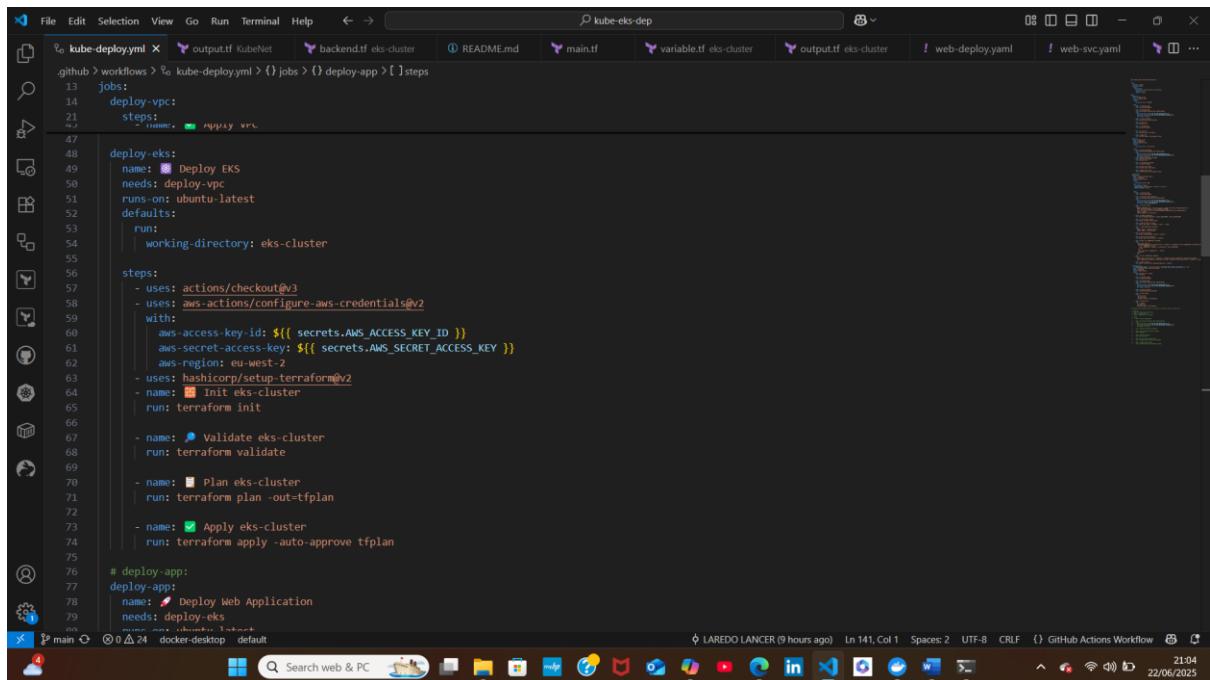
© 2025 Gianik Healthcare Solutions. All Rights Reserved.

19:45
22/06/2025

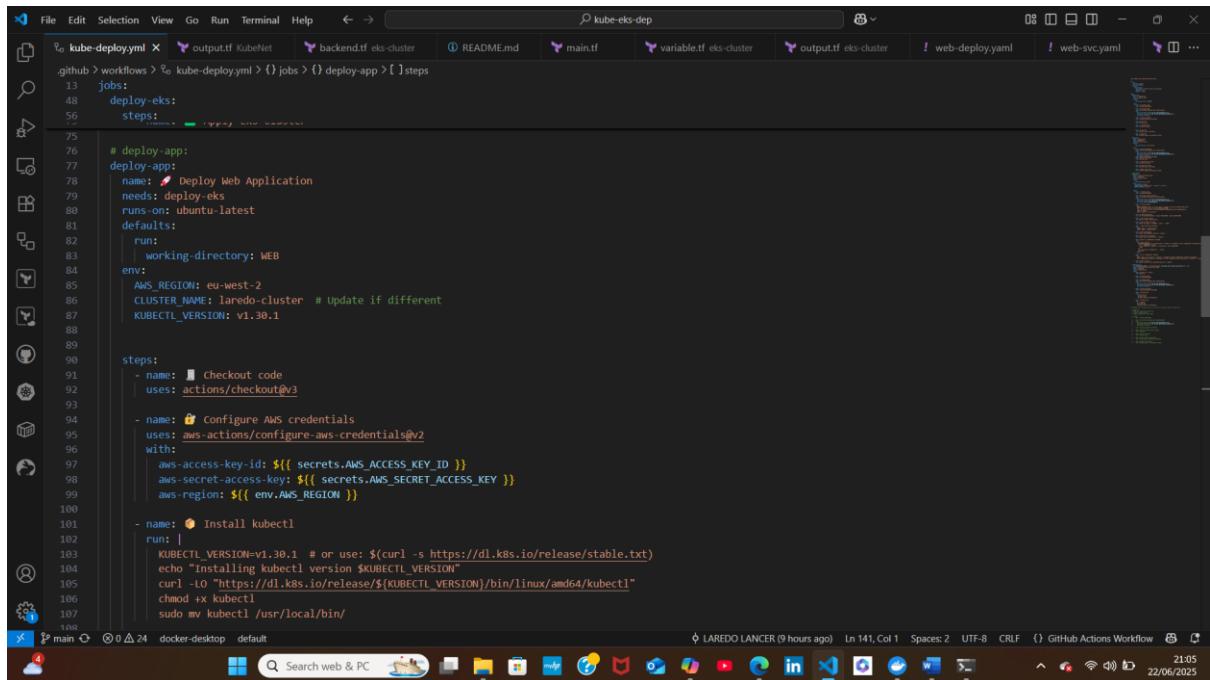
GitHub Actions Workflow



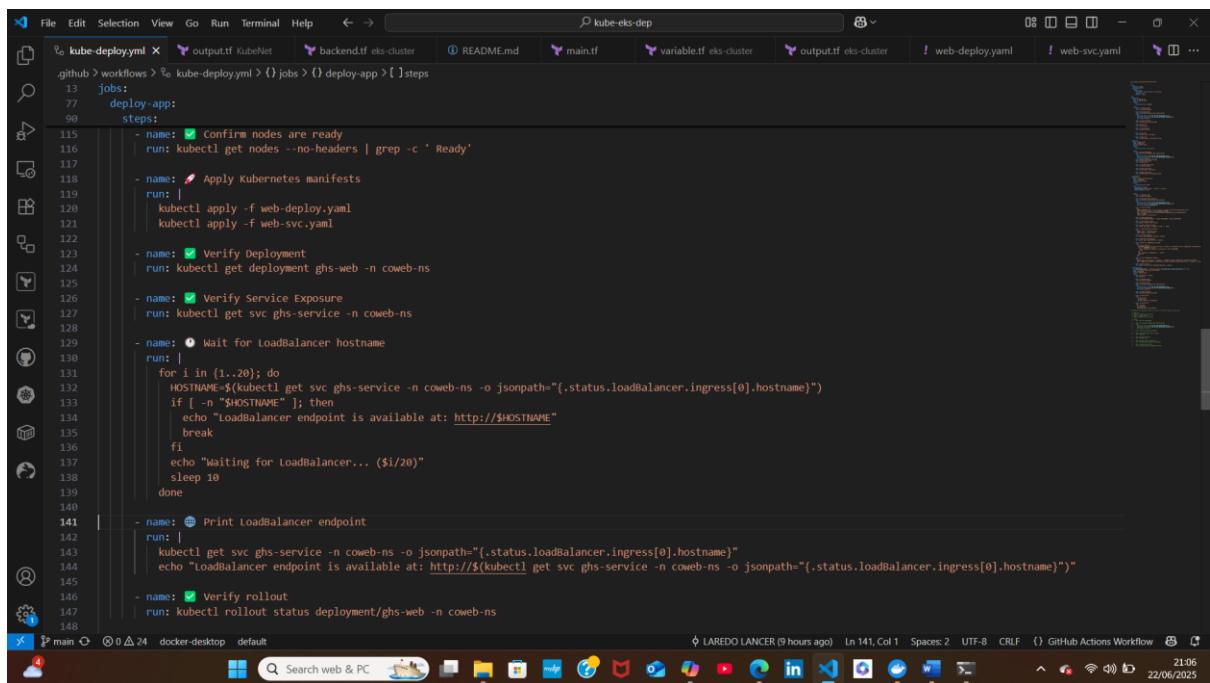
```
File Edit Selection View Go Run Terminal Help ← → kube-eks-dep README.md main.tf variable.tf eks-cluster output.tf eks-cluster web-deploy.yaml web-svc.yaml backend.tf KubeNet variable.tf KubeNet
.github > workflows > kube-deployment > jobs > deploy-app > steps
1 name: Deploy and Clean EKS Infrastructure
2
3 on:
4   push:
5     branches: [main]
6   workflow_dispatch:
7     inputs:
8       auto_destroy:
9         description: 'Destroy Infra? (true/false)'
10        required: true
11        default: 'false'
12
13 jobs:
14   deploy-vpc:
15     name: 🛠 Deploy VPC
16     runs-on: ubuntu-latest
17     defaults:
18       run:
19         working-directory: KubeNet
20
21     steps:
22       - name: 🛡 Checkout Code
23         uses: actions/checkout@v3
24
25       - name: 🔑 Configure AWS
26         uses: aws-actions/configure-aws-credentials@v2
27         with:
28           aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
29           aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
30           aws-region: eu-west-2
31
32       - name: 🛠 Setup Terraform
33         uses: hashicorp/setup-terraform@v2
34
35       - name: 🛡 Init VPC
36         run: terraform init
37
38       - name: 🔍 Validate VPC
39         run: terraform validate
40
41       - name: 🛡 Plan VPC
42         run: terraform plan -out=tfplan
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
```



```
File Edit Selection View Go Run Terminal Help ← → kube-eks-dep README.md main.tf variable.tf eks-cluster output.tf eks-cluster web-deploy.yaml web-svc.yaml backend.tf KubeNet variable.tf KubeNet
.github > workflows > kube-deployment > jobs > deploy-app > steps
13 jobs:
14   deploy-vpc:
15     steps:
16       - name: 🛡 Apply VPC
17
18
19
20
21   deploy-eks:
22     name: 🛠 Deploy EKS
23     needs: deploy-vpc
24     runs-on: ubuntu-latest
25     defaults:
26       run:
27         working-directory: eks-cluster
28
29     steps:
30       - uses: actions/checkout@v3
31       - uses: aws-actions/configure-aws-credentials@v2
32         with:
33           aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
34           aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
35           aws-region: eu-west-2
36       - uses: hashicorp/setup-terraform@v2
37       - name: 🛡 Init eks-cluster
38         run: terraform init
39
40       - name: 🔍 Validate eks-cluster
41         run: terraform validate
42
43       - name: 🛡 Plan eks-cluster
44         run: terraform plan -out=tfplan
45
46       - name: 🛡 Apply eks-cluster
47         run: terraform apply --auto-approve tfplan
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
```



```
.github > workflows > %o kube-deploy.yml > () jobs > () deploy-app > [ ] steps
13   jobs:
14     deploy-eks:
15       steps:
16         - name: 📁 Checkout code
17           uses: actions/checkout@v3
18
19         - name: 🔑 Configure AWS credentials
20           uses: aws-actions/configure-aws-credentials@v2
21           with:
22             aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
23             aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
24             aws-region: ${{ env.AWS_REGION }}
25
26         - name: ⚒ Install kubectl
27           run:
28             - KUBECTL_VERSION=v1.30.1 # or use: $(curl -s https://dl.k8s.io/release/stable.txt)
29             - echo "Installing kubectl version $KUBECTL_VERSION"
30             - curl -Lo "https://dl.k8s.io/release/${KUBECTL_VERSION}/bin/linux/amd64/kubectl"
31             - chmod +x kubectl
32             - sudo mv kubectl /usr/local/bin/
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
509
```



```
.github > workflows > %o kube-deploy.yml > () jobs > () deploy-app > [ ] steps
13   jobs:
14     deploy-app:
15       steps:
16         - name: 📁 Confirm nodes are ready
17           run: kubectl get nodes --no-headers | grep -c ' Ready'
18
19         - name: 🔑 Apply Kubernetes manifests
20           run:
21             - kubectl apply -f web-deploy.yaml
22             - kubectl apply -f web-svc.yaml
23
24         - name: ✅ Verify Deployment
25           run: kubectl get deployment ghs-web -n coweb-ns
26
27         - name: ✅ Verify Service Exposure
28           run: kubectl get svc ghs-service -n coweb-ns
29
30         - name: ⚒ Wait for LoadBalancer hostname
31           run:
32             - for i in {1..20}; do
33               HOSTNAME=$(kubectl get svc ghs-service -n coweb-ns -o jsonpath=".status.loadBalancer.ingress[0].hostname")
34               if [ -n "$HOSTNAME" ]; then
35                 echo "LoadBalancer endpoint is available at: http://$HOSTNAME"
36                 break
37               fi
38               echo "Waiting for LoadBalancer... ($i/20)"
39               sleep 10
40             done
41
42         - name: 🌐 Print LoadBalancer endpoint
43           run:
44             - kubectl get svc ghs-service -n coweb-ns -o jsonpath=".status.loadBalancer.ingress[0].hostname"
45             - echo "LoadBalancer endpoint is available at: http://$HOSTNAME"
46
47         - name: ✅ Verify rollout
48           run: kubectl rollout status deployment/ghs-web -n coweb-ns
49
50
51
52
53
54
55
56
57
58
59
59
```

A screenshot of a GitHub Actions workflow file named `kube-deploy.yml` in a dark-themed code editor. The file contains YAML code defining a workflow with jobs for deployment and teardown. The code includes steps for checking out code, configuring AWS, setting up Terraform, and destroying EKS and VPC resources. The GitHub Actions interface is visible at the bottom of the screen.

```

jobs:
  deploy-app:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v3
      - name: Configure AWS
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: eu-west-2
      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v2
      - name: Destroy EKS
        run: |
          cd eks-cluster
          terraform init
          terraform destroy -auto-approve
      - name: Destroy VPC
        run: |
          cd ../KubeNet
          terraform init
          terraform destroy -auto-approve

```

GitHub Actions Pipeline

A screenshot of the GitHub Actions pipeline interface for the workflow `kube-eks-dep`. The pipeline has completed successfully, indicated by a green checkmark icon and the text "Revaluated 106 #89". The "Deploy VPC" job is highlighted, showing its status as "succeeded 3 minutes ago in 31s". The pipeline summary shows other jobs: Deploy VPC (green), Deploy EKS (green), Deploy Web Application (green), and Teardown Infra (auto after del..) (grey). The GitHub Actions interface is visible at the bottom of the screen.

Teardown of Infrastructure

The screenshot shows a GitHub Actions workflow run for a repository named 'Oreire / kube-eks-dep'. The workflow has four jobs: 'Deploy VPC' (green), 'Deploy EKS' (green), 'Deploy Web Application' (green), and 'Teardown Infra (auto after delay)' (yellow). The 'Teardown Infra' job is currently running. The logs for this job show the following steps:

```
✓ Teardown Infra (auto after delay)
Started 10s ago
  • Set up job
  • Wait for 2 minutes
  • Checkout Code
  • Configure AWS
  • Setup Terraform
  • Destroy EKS
  • Destroy VPC
```

The workflow was started 10 seconds ago. The 'Destroy EKS' step is currently in progress, indicated by a yellow circle icon.

This screenshot shows the same GitHub Actions workflow run as the previous one, but with the logs for the 'Teardown Infra' job expanded. The expanded logs show the Terraform initialization process:

```
1 ► Run cd eks-cluster
12 /home/runner/work/_temp/f0fc7e64-d50f-4aa7-a2d4-30b37f5b701/terraform-bin init
13 Initializing the backend...
14
15 Successfully configured the backend "s3"! Terraform will automatically
16 use this backend unless the backend configuration changes.
17 Initializing provider plugins...
18 - terraform.io/builtin/terraform is built in to Terraform
19 - Finding latest version of hashicorp/aws...
20 - Installing hashicorp/aws v6.0.0...
21 - Installed hashicorp/aws v6.0.0 (signed by HashiCorp)
22 Terraform has created a lock file .terraform.lock.hcl to record the provider
23 selection it made above. Include this file in your version control repository
24 so that Terraform can guarantee to make the same selections by default when
25 you run "terraform init" in the future.
26 Terraform has been successfully initialized!
27
28 You may now begin working with Terraform. Try running "terraform plan" to see
29 any changes that are required for your infrastructure. All Terraform commands
30 should now work.
```

The logs indicate that Terraform has successfully initialized the AWS provider and created a lock file.

Conclusion

Deployed a production-grade EKS architecture with Terraform and GitHub Actions to provision and manage a resilient Kubernetes environment on AWS. The solution included modular Terraform layers, automated application deployment, and integrated teardown workflows for full infrastructure lifecycle management.

Key Challenges and Lessons Learned

Although the deployment of this production-grade EKS architecture was ultimately successful—thanks to thoughtful IAM planning, integrated teardown workflows, and a modular design—it required addressing several critical challenges along the way before achieving a truly stable, secure, and scalable infrastructure.

Below is a breakdown of key problem areas to watch for:

1. Network Complexity and Isolation

Managing the interplay between public and private subnets, NAT Gateways, security groups, and potential service mesh boundaries added significant complexity. Without precise configuration, the infrastructure risked misrouted traffic or unintended exposure of internal services.

2. IAM Role Misconfigurations

EKS heavily depends on tightly scoped IAM roles for control plane access, node group operations, and workload-level permissions. Seemingly minor missteps—such as omitting `eks:DescribeUpdate` or `iam:CreateRole`—caused deployment errors and blocked critical updates, prompting a revisit of role boundaries and trust relationships.

3. Bootstrap Failures in Node Groups

Some nodes initially failed to join the cluster due to incorrect IAM roles, improperly tagged subnets, or missing bootstrap configurations. Troubleshooting these issues emphasized the importance of validating node IAM roles, subnet reachability to the EKS endpoint, and proper user data scripts.

4. Terraform State Drift and Partial Teardowns

Orphaned infrastructure—such as lingering ENIs, EIPs, or route table associations—caused ‘DependencyViolation’ errors during teardown. Ensuring logical deletion order, proper resource dependencies, and explicit lifecycle rules proved essential for clean and repeatable destruction.

5. Public LoadBalancer Exposure

While deploying services via an external LoadBalancer accelerated access, it also introduced potential security blind spots. To mitigate exposure risks, additional layers such as Kubernetes Network Policies, WAF integrations, and Ingress Controllers were identified as future hardening steps.

6. Update Collisions

Parallel update attempts on the EKS cluster often led to ‘ResourceInUseException’ errors, particularly during overlapping Terraform apply events. Gating mechanisms, status polling, and sequential update strategies were introduced to prevent mid-flight configuration conflicts.

7. Observability Gaps

In the initial deployment, visibility into cluster health was limited. The absence of centralized logging and monitoring delayed response to performance issues. Integrating tools like Fluent Bit for logs and Prometheus/Grafana for metrics became a priority to reduce mean time to resolution (MTTR) and enhance operational insight.

Outcome

Through systematic troubleshooting and iterative refinement of IAM roles, network topology, Terraform lifecycles, and deployment workflows, I delivered a stable, secure, and fully automated EKS architecture. The platform now supports GitOps-style delivery, modular provisioning, and clean teardown—positioned for further enhancements such as ArgoCD, secrets management, and workload observability.

Next steps:

1. Strengthening IAM Governance

- Refactor IAM roles using least-privilege principles with scoped permissions for provisioning, deployment, and teardown stages.
- Use tools like **AWS IAM Access Analyzer** or **Policy Sentry** to detect overly permissive policies.

2. Improved Observability

- Deploy **Fluent Bit** to forward cluster-wide logs to CloudWatch or an ELK stack.
- Set up **Prometheus** and **Grafana** for monitoring EKS health and application performance.
- Define SLOs for latency, availability, and error rates.

3. Enforcing CI/CD Validation

- Add pre-apply checks in GitHub Actions to validate:
 - Ongoing EKS updates
 - Terraform drift and plan output
- Integrate **OPA Gatekeeper** or **Polaris** for policy enforcement on manifests.

4. Adoption of ArgoCD for GitOps

- Transition from kubectl apply to **ArgoCD-managed deployments** using Kustomize overlays.
- Enable automated sync, pruning, and self-healing to maintain state consistency.

5. Improvements of Teardown Lifecycle

- Use tags or workspace identifiers to distinguish environments.
- Add pre-destroy checks for dangling resources like ENIs.
- Optionally implement TTL-based auto-cleanup using scheduled GitHub Actions.

6. Secured Network Boundaries

- Replacement of LoadBalancer services (AWS NLB@L3) with an **Ingress controller** (e.g., NGINX or AWS ALB) @ L7 of the OSI layer
- Enable Web Application Firewall (**WAF**) protection for public endpoints.
- Apply **Network Policies** to control inter-namespace traffic.

7. Centralization of Secrets Management

- Migrate to **AWS Secrets Manager**, **Parameter Store**, or **Sealed Secrets**.
- Enforce regular secret rotation and limit access via KMS policies and RBAC.